**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)

|  |  |
| --- | --- |
| Roll No. : S020 | Name: Husain Chhil |
| Class : MBA tech (Data Science) | Batch : J1 |
| Date of Experiment : | Date/Time of Submission : |
| Grade : |  |

**B.1 Code:**

**Models.py**

from sklearn.datasets import fetch\_openml

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

import joblib

glass = fetch\_openml(name = "glass", version = 1)

X = glass.data

y = glass.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = RandomForestClassifier(n\_estimators = 100, random\_state = 42)

model.fit(X\_train, y\_train)

joblib.dump(model, 'glass\_classification\_model.pkl')

**Views.py**

from django.shortcuts import render

from .forms import GlassPredictionForm

import joblib

def predict\_glass\_type(request):

    prediction = None

    if request.method == 'POST':

        form = GlassPredictionForm(request.POST)

        if form.is\_valid():

            model = joblib.load('glass\_classification\_model.pkl')

            features = form.get\_features\_as\_array()

            prediction = model.predict(features)[0]

    else:

        form = GlassPredictionForm()

    return render(request, 'Prediction.html', {'form': form, 'prediction': prediction})

**Forms.py**from django import forms

import numpy as np

class GlassPredictionForm(forms.Form):

feature\_1 = forms.FloatField(label="RI")

feature\_2 = forms.FloatField(label="Na")

feature\_3 = forms.FloatField(label="Mg")

feature\_4 = forms.FloatField(label="Al")

feature\_5 = forms.FloatField(label="Si")

feature\_6 = forms.FloatField(label="K")

feature\_7 = forms.FloatField(label="Ca")

feature\_8 = forms.FloatField(label="Ba")

feature\_9 = forms.FloatField(label="Fe")

def get\_features\_as\_array(self):

# Manually collect values from the form fields

features = np.array([

self.cleaned\_data['feature\_1'],

self.cleaned\_data['feature\_2'],

self.cleaned\_data['feature\_3'],

self.cleaned\_data['feature\_4'],

self.cleaned\_data['feature\_5'],

self.cleaned\_data['feature\_6'],

self.cleaned\_data['feature\_7'],

self.cleaned\_data['feature\_8'],

self.cleaned\_data['feature\_9']

])

return features.reshape(1, -1)

**Prediction.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Glass Type Prediction</title>

    <style>

        body {

            font-family: Arial, sans-serif;

        }

        .container {

            max-width: 900px;

            margin: 0 auto;

            padding: 20px;

            display: flex;

            gap: 20px;

        }

        .form-container, .result-container {

            flex: 1;

        }

        form {

            display: grid;

            gap: 10px;

        }

        label {

            font-weight: bold;

        }

        .result-container {

            padding: 20px;

            background-color: #f0f0f0;

            border: 1px solid #ccc;

        }

    </style>

</head>

<body>

    <div class="container">

        <div class="form-container">

            <h1>Glass Type Prediction</h1>

            <form method="POST">

                {% csrf\_token %}

                {{ form.as\_p }}

                <div>

                    <button type="submit">Predict</button>

                </div>

            </form>

        </div>

        <div class="result-container">

            {% if prediction %}

            <h2>Prediction Result:</h2>

            <p>The predicted glass type is: <strong>{{ prediction }}</strong></p>

            {% else %}

            <p>Enter values and click Predict to see the result.</p>

            {% endif %}

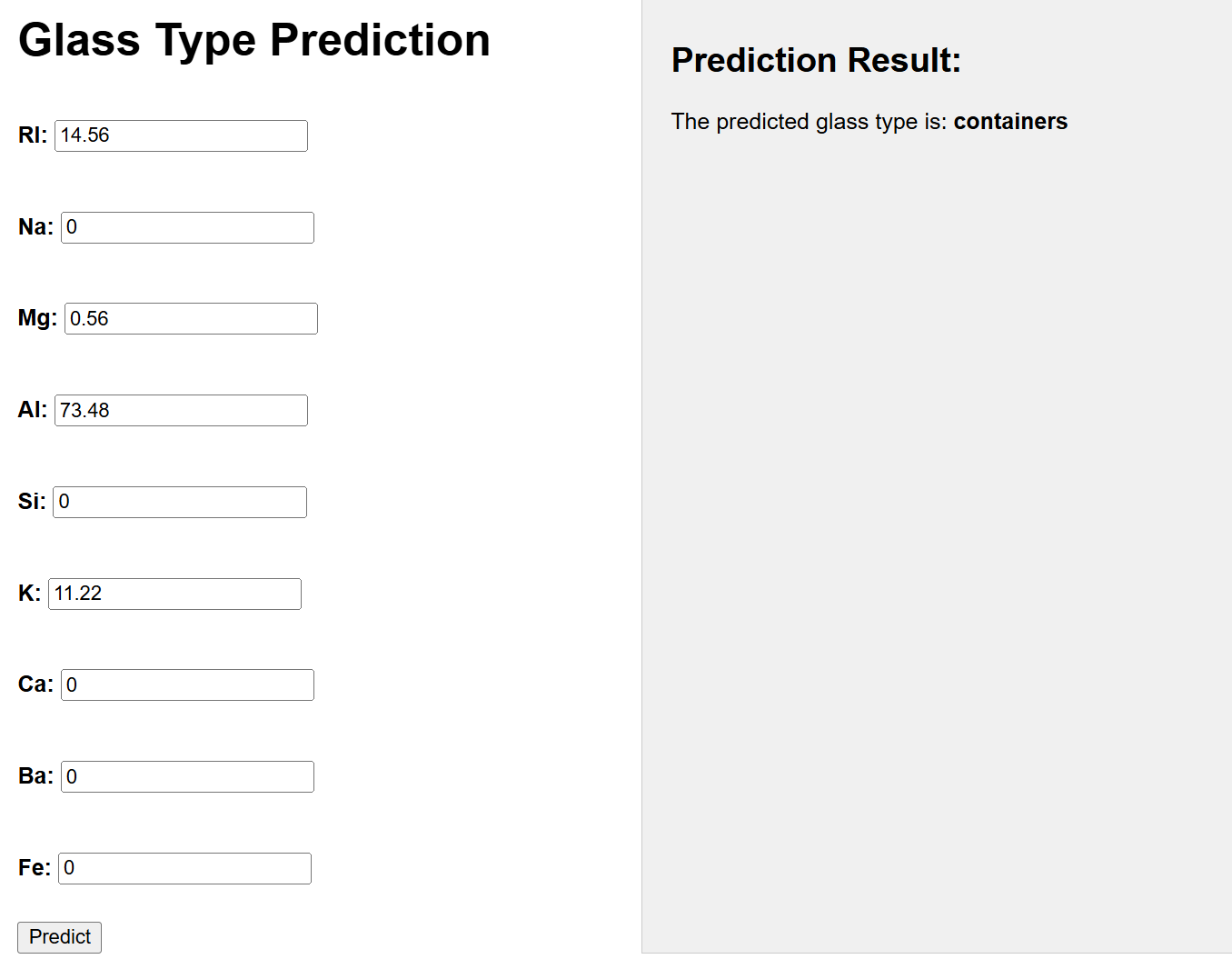
        </div>

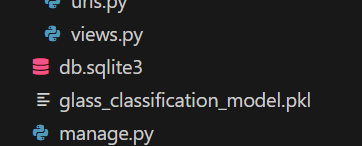
    </div>

</body>

</html>

**B.2 Output**

****

****

**B.3 Conclusion:**

We learnt to make a reusable Django form that can gather the required features for model prediction. This form can be used in a Django view to handle user inputs, pass them to the trained model, and display the classification results, providing a fully interactive experience for the user.

**B.4 Question of Curiosity**

1. How does Django handle form data for making machine learning predictions?

Ans: Django uses forms.Form classes to define fields that accept user input and validate it. In a machine learning application, we create a form with fields corresponding to model input features. When the user submits the form, Django collects and cleans the data. We can then convert this cleaned data into the appropriate format (e.g., a NumPy array or Pandas DataFrame) and pass it to the machine learning model for prediction.

1. How can you save and load a machine learning model in Django?

Ans: To save and load machine learning models in Django, you can use libraries like joblib or pickle. Here’s how:

* **Saving**: After training your model, use joblib.dump(model, 'model.pkl') to save it to a file (e.g., in the models/ directory).
* **Loading**: Load the model in your Django view with model = joblib.load('path/to/model.pkl'). This allows you to reuse the trained model without retraining it each time.

1. What are the steps involved in integrating a machine learning model with a Django web application?

Ans:

 **Train and Save the Model**: Train your machine learning model in a Jupyter notebook or script, then save it as a .pkl or .joblib file.

 **Create a Django Form**: Define a form with fields corresponding to the input features of your model to capture user input.

 **Build a View for Prediction**: Load the saved model in the view, get the cleaned form data, and use it as input for the model's predict function.

 **Display the Result**: Pass the prediction result from the model to the HTML template to display it to the user.

 **Front-End**: Design the form and result display in a Django HTML template, and make the user experience interactive.

1. What are some advantages of deploying a machine learning model using Django?

Ans:

 **Full-Stack Framework**: Django provides a complete suite of tools to handle the front end, back end, and database, allowing for end-to-end deployment.

 **User-Friendly Interface**: With Django’s form and template system, you can easily create user-friendly interfaces for non-technical users to interact with the model.

 **Built-In Security**: Django provides built-in security features (e.g., CSRF protection, input sanitization) to ensure safe handling of user input.

 **Scalability**: Django supports horizontal scaling, which allows your application to handle high loads by adding more servers as needed.

 **Ecosystem Support**: Django integrates well with other technologies and libraries for tasks such as model serialization, caching, and API creation (e.g., Django REST Framework).

1. How does Django handle user input security in the context of machine learning applications?

Ans: Django has several built-in mechanisms for handling user input securely:

* **CSRF Protection**: Django uses Cross-Site Request Forgery protection for form submissions, reducing the risk of malicious requests.
* **Input Validation and Sanitization**: Django forms automatically validate and sanitize input data, which helps prevent SQL injection and other attacks.
* **XSS Protection**: Django escapes any output rendered in templates, protecting against Cross-Site Scripting (XSS) attacks.
* **Field Type Enforcement**: When defining forms, Django enforces the data type (e.g., FloatField, IntegerField), ensuring that only valid data is processed.

1. What challenges might you face when deploying machine learning models with Django in a production environment?

Ans:

 **Scalability**: Machine learning models can be resource-intensive, and handling multiple requests in real time may lead to performance bottlenecks.

 **Latency**: Depending on the complexity of the model, generating predictions can take time, which may lead to delays in a live application.

 **Dependency Management**: ML models often require specific library versions, which can conflict with Django’s dependencies.

 **Model Retraining**: Keeping the model updated with new data requires a system for regular retraining and deployment.

 **Security**: Handling sensitive data securely is crucial, especially if the model uses user data for predictions.

 **Resource Allocation**: Handling resource-heavy models (like deep learning models) may require specialized infrastructure (e.g., GPUs or distributed systems).

1. What are some best practices for deploying machine learning models in a Django application?

Ans:

 **Use a Separate API for Predictions**: Consider using Django REST Framework to create an API endpoint for predictions, allowing you to decouple the model from the web interface.

 **Optimize the Model**: Use optimized versions of your model, such as pruning or quantization, to reduce latency.

 **Implement Caching**: Cache frequently requested predictions to reduce computation time and enhance performance.

 **Containerization**: Use Docker to encapsulate dependencies and provide consistent environments across development and production.

 **Monitor and Log Predictions**: Track prediction requests, inputs, and outputs to understand usage patterns and catch any issues early.

 **Automate Retraining and Deployment**: If your model needs to be updated regularly, set up a pipeline for model retraining, validation, and deployment.

 **Use Cloud Services if Needed**: For complex models, consider deploying on cloud services with scalable compute resources (e.g., AWS SageMaker or Google AI Platform) and connecting them to your Django application.